



Clase 12. Python

# ***Programación Orientada a Objetos - I***

***RECUERDA PONER A GRABAR LA  
CLASE***





## ***OBJETIVOS DE LA CLASE***

- Conocer qué es la POO.
- Diferenciar la POO de la programación tradicional.
- Aplicar el pensamiento computacional en un programa con objetos.

# ***CRONOGRAMA DEL CURSO***

## Clase 11



### Excepciones



DESAFÍO DE ERRORES



DESAFÍO DE EXCEPCIONES

## Clase 12



### Clases y objetos



MI PRIMER PENSAMIENTO EN  
POO



NUESTRO PRIMER DC



UTILIZAMOS PLANTTEXT

## Clase 13



### Clases y objetos 2



MI PRIMERA CLASE



OLIMPIADAS

***¿QUÉ ERA LA POO?***



# *¿Qué es la POO?*

Es un modo o paradigma de programación, que nos permite organizar el código pensando el problema como una relación entre “cosas”, denominadas objetos. Los objetos se trabajan utilizando las “clases”. Estas nos permiten agrupar un conjunto de variables y funciones que veremos a continuación.

# ***MOTIVACIÓN***

# ***Motivación***

Los programadores se han dedicado a construir aplicaciones muy parecidas que resolvían una y otra vez los mismos problemas. Para conseguir que los esfuerzos de los programadores puedan ser reutilizados se creó la posibilidad de utilizar módulos. El primer módulo existente fue la función.

**Pero la función se centra más en aportar una funcionalidad dada, pero no tiene tanto interés con los datos.**





# ***Motivación***

Con la POO se busca resolver aplicaciones cada vez más complejas, sin que el código se vuelva un caos. Además, se pretende dar pautas para realizar las cosas de manera que otras personas puedan utilizarlas y adelantar su trabajo, de manera que consigamos que el código se pueda reutilizar.





## ***RESUMIENDO***

Lo importante de la POO es poder separar los problemas generales en suma de pequeños problemas aislados, para poder modelar la solución y en un futuro que cualquiera pueda utilizar varios de estos módulos creados por este paradigma.

# ***DIFERENCIAS CON LA PROGRAMACIÓN TRADICIONAL***



# ***¿Qué es un paradigma de programación?***

Un paradigma de programación es un estilo de desarrollo de programas.

Es decir, **un modelo para resolver problemas computacionales.**

Los lenguajes de programación se encuadran en uno o varios paradigmas a la vez a partir del tipo de órdenes que permiten implementar, algo que tiene una relación directa con su sintaxis.



# ***PARADIGMAS DE PROGRAMACIÓN***

- **Imperativo:** Los programas se componen de un conjunto de sentencias que cambian su estado. Son secuencias de comandos que ordenan acciones a la computadora.
- **Declarativo:** Opuesto al imperativo. Los programas describen los resultados esperados sin listar explícitamente los pasos a llevar a cabo para alcanzarlos.
- **Lógico:** El problema se modela con enunciados de lógica del primer orden.



# ***PARADIGMAS DE PROGRAMACIÓN***

- **Funcional:** Los programas se componen de funciones, es decir, implementaciones de comportamiento que reciben un conjunto de datos de entrada y devuelven un valor de salida.
- **Orientado a Objetos:** El comportamiento del programa es llevado a cabo por objetos, entidades que representan elementos del problema a resolver y tienen atributos y comportamiento.

# ***Diferencias con la programación tradicional***

## Programación Estructurada



## Programación Orientada a Objetos

- Énfasis en la transformación de datos.
- Las funciones y los datos son manejados separadas.
- **Difícil de entender y modificar.**

- Énfasis en la abstracción de datos.
- Las funciones y los datos son encapsulados en una entidad como entidades.
- **Facilita su mantenimiento y comprensión es orientada al mundo real.**

# ***VENTAJAS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS***





# PROGRAMACIÓN ORIENTADA A OBJETOS

## ***VENTAJAS***

- **No se encapsulan los atributos**, ni el acceso a los atributos desde las funciones, que también están encapsuladas dentro de la clase.
- **Ofrece la posibilidad de heredar de unas clases a otras** para que puedan acceder a los miembros de la clase padre, con lo cual podemos solucionar de una manera más eficiente, el ampliar el comportamiento de nuestros objetos o clases.



## PROGRAMACIÓN ORIENTADA A OBJETOS

# ***VENTAJAS***

- **Podemos hacernos visualmente una idea más clara de cómo se puede comportar la clase**, además de tener en el mismo sitio tanto las funciones que hacen que podamos manejar la clase como los datos que vamos a manejar, etc.



***BREAK***

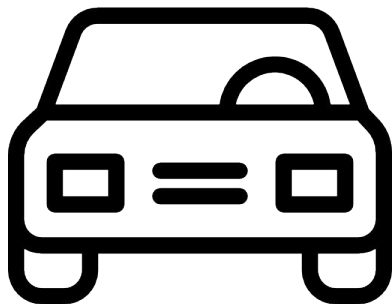
**¡5/10 MINUTOS Y VOLVEMOS!**

***¿CÓMO SE PIENSA CON POO?***

# ***¿Cómo se piensa con POO?***

Es muy parecido a cómo lo haríamos en la vida real.

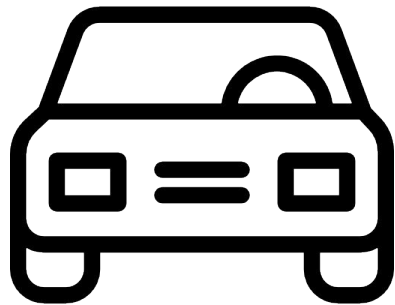
Por ejemplo vamos a pensar en un coche para tratar de **modelizar en un esquema de POO**.



- **Elemento principal:** coche.
- **Características:** marca, modelo, color, etc.
- **Funcionalidades:** reversa, aparcamiento, etc.

# ***¿Cómo se piensa con POO?***

Ahora si pensamos el ejemplo anterior desde el esquema de Programación Orientada a Objetos, sería de la siguiente manera:



Elemento Principal → **Clase**

Características → **Atributos**

Funcionalidades → **Métodos**

# ¿Cómo se piensa con POO?

Para poder comentar el anterior esquema de POO fácilmente aparecen los denominados **diagramas de clases**, que ilustran todo según estándares de la UML.



Estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, operaciones, y las relaciones entre los objetos

# ***Ejemplos de diagramas de clases***

Persona
- nombre - edad
+ mostrar()

Pelicula
+Nombre +Tipo +Hora +Sinopsis +idioma +clasificacion +ID_Responsable_Taquilla
+ver_trailer() +Seleccionar_Horario() +Seleccionar_pelicula() +salir()

Vuelo
- numero de vuelo : int - hora de salida : String - hora de llegada : String - lugar de origen : String - lugar de destino : String - fecha de salida : Date - fecha de regreso : Date - tipo de vuelo : String
+ reservar () : void + modificar () : void + eliminar () : void + buscar () : Vuelo





# ***MI PRIMER PENSAMIENTO CON POO***

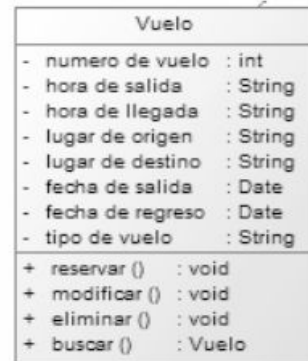
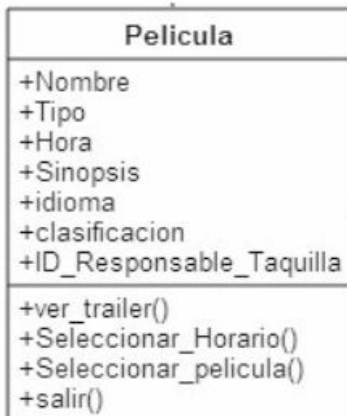
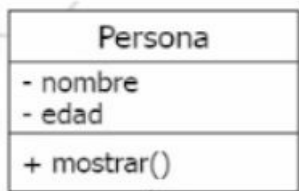
¿Cómo sería el problema a resolver que se modela con las siguientes clases?

**Tiempo Estimado:** 10 minutos.



# ***Mi primer pensamiento con POO***

¿Cómo sería el problema a resolver que se modela con las siguientes clases?. Utilizando los diagramas que vimos recién: ¿cuál sería el problema que se quiere resolver?. ¿Hay o habría una relación entre las clases?.



**Tiempo Estimado:** 10 minutos.



Con el ejercicio anterior notamos que los problemas complejos no son sólo una suma de Clases, son además **relaciones entre clases.**

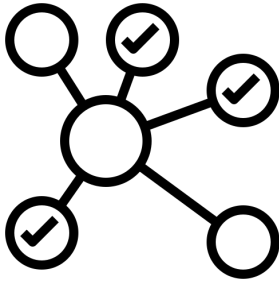
# ***Relaciones entre clases***

# ***TIPOS DE RELACIONES***

# ***Tipos de relaciones***

La Orientación a Objetos (POO) intenta modelar aplicaciones del mundo real tan fielmente como sea posible y por lo tanto **debe reflejar estas relaciones entre clases y objetos.**

**Existen tres grandes tipos de relaciones:**

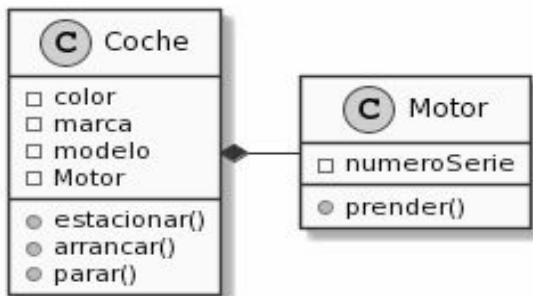
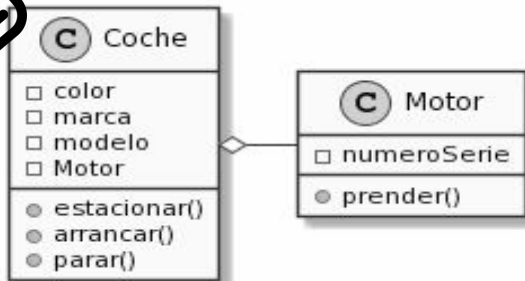


- Agregación / Composición
- Asociación
- Generalización / Especialización.

Herencia simple y herencia múltiple.

# Agregación

EXAMPLE



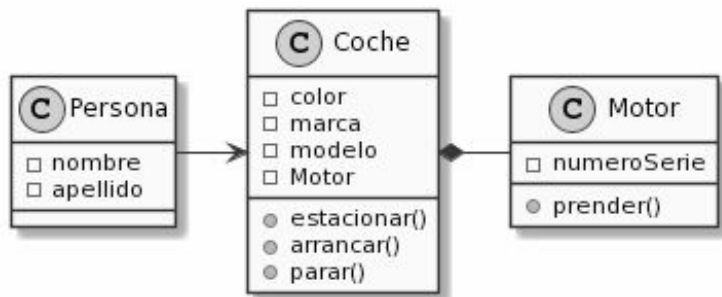
Esta relación se presenta entre una clase **TODO** y una clase **PARTE** que es **componente de TODO**.

La implementación de este tipo de relación se consigue definiendo como atributo un objeto de la otra clase que es parte-de.

Los objetos de la clase **TODO** son objetos contenedores. **Un objeto contenedor es aquel que contiene otros objetos.**

# Asociación

EXAMPLE



Especifica una relación semántica entre **objetos no relacionados**. Este tipo de relaciones permiten crear asociaciones que capturen los participantes en una relación semántica.

Son relaciones del tipo **“pertenece\_a”** o **“está asociado\_con”**.

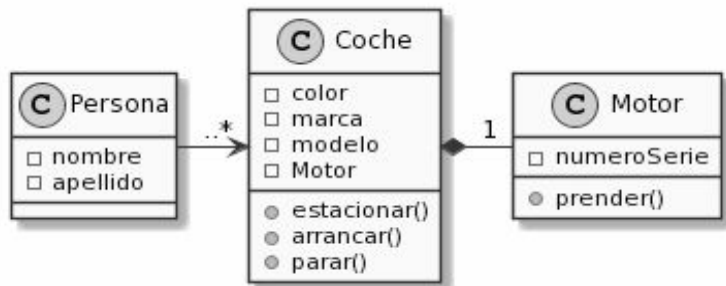
Se da cuando una clase usa a otra clase para realizar algo.



# ***CARDINALIDAD DE LAS RELACIONES***

# Cardinalidad de las relaciones

EXAMPLE

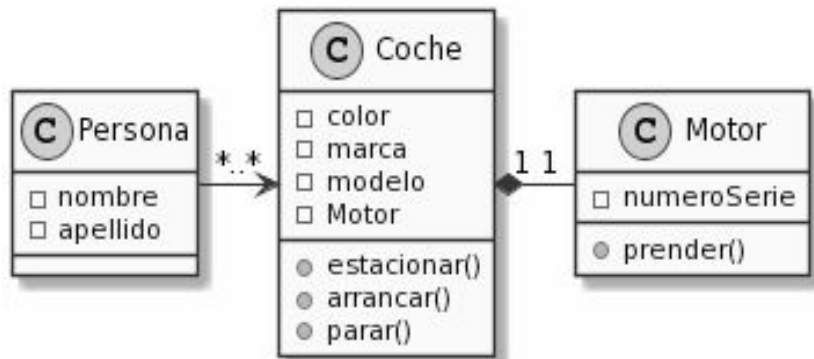


Sabemos que un coche tiene un motor, y que la persona está asociada a un vehículo.

Ahí nace el **concepto de cardinalidad, es decir indicar el número de Objetos que están en la relación.** Por ejemplo, una persona puede tener muchos coches, y que los coches solo tienen un motor.

# Cardinalidad de las relaciones

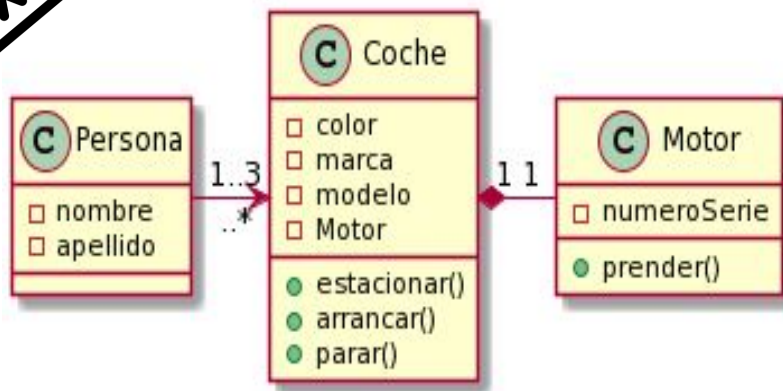
**EXAMPLE**



Además, sabemos que cada motor pertenece a un solo coche, y los coches a veces tienen más de un titular que lo usan, es decir tienen a varias personas para usarlo.

# Cardinalidad de las relaciones

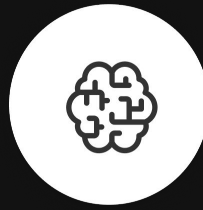
EXAMPLE



Solo por completitud del tema, y un poco alejado de la realidad, supongamos que un coche solo puede tener entre 1 y 3 titulares.



En la cardinalidad, lo importante es marcar si la relación es a muchos objetos (“..\*”) o si solo es a uno (**1**), incluso a veces se puede decir a uno o ninguno (**0,1**).



***¡PARA PENSAR!***

*Con lo aprendido hasta el momento en la clase están listos  
para responder ¿qué son los diagramas de clases?*

RESPONDAN EN EL CHAT DE ZOOM

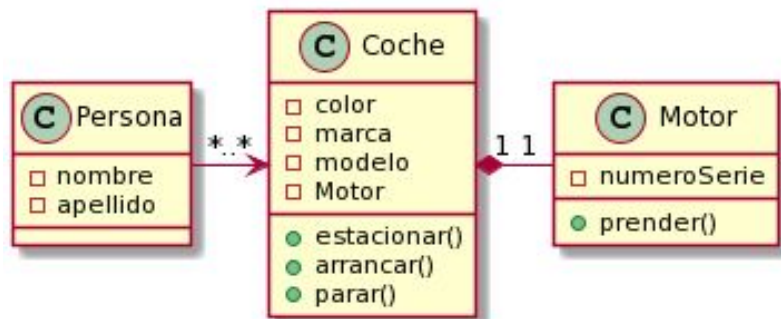
# ***DIAGRAMAS DE CLASES***

# ¿Qué son?

Podemos definir a los diagramas de clases como **las clases que usaremos en el programa con las relaciones entre sí.**

*La siguiente imagen representa un diagrama de clases:*

**EXAMPLE**





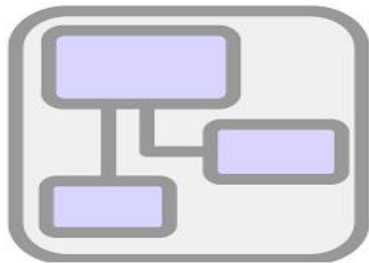
# ***GRAFICAR DIAGRAMAS DE CLASES***



# ***Graficar diagrama de Clases***

Hay múltiples programas que nos permiten realizar los diagramas, los más populares son [Día](#), [Visio](#) y algunos otros online, como [Moqups](#), u [Online Visual Paradigm](#).

Nosotros usaremos [PlantText](#), una web que nos permite hacer de manera ágil y sencilla la estructuración de los diagramas.



## PlantText

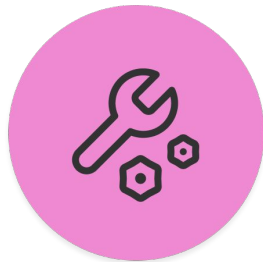
The expert's design tool

***CODER HOUSE***



## ***EJEMPLO EN VIVO***

*Vamos a ingresar a [PlantText](#) y aprenderemos cómo usarlo.*



## ***NUESTRO PRIMER DC***

Crear un diagrama de clases utilizando Planttext.

*Tiempo Estimado: 15 minutos.*



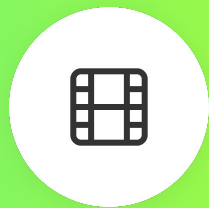
## ***Nuestro primer DC***

Crear un diagrama de clases con PlantText para modelar la siguiente situación problemática: Se quiere crear un programa para administrar los **productos** que se venden en un **negocio**, el negocio tiene varios **empleados** y cada empleado posee un **contacto**.

**Tiempo Estimado:** 15 minutos.

***¿PREGUNTAS?***





***¿QUIERES SABER MÁS? TE DEJAMOS  
MATERIAL AMPLIADO DE LA CLASE***



EjemploClase

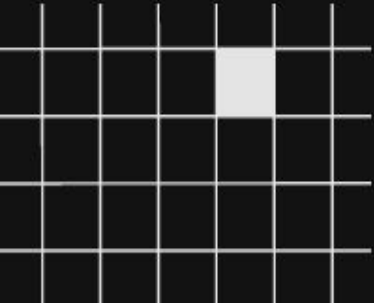






# ***¡MUCHAS GRACIAS!***

Resumen de lo visto en clase hoy:

- Programación Orientada a Objetos.
  - Relaciones entre clases.
  - Diagrama de clases.
  - Diferencia entre POO y Tradicional.
  - Ventajas de la POO.
- 



***OPINA Y VALORA ESTA CLASE***

***#DEMOCRATIZANDO LA EDUCACIÓN***

***CODER HOUSE***